VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

WEATHER FORECAST USING DATA MINING TECHNIQUES

CSE3019 – DATA MINING

J Component – Final Review - 3

BATCH NO.: 11

NAME: 1) KRUPAL J. KATHROTIA (17BCE0589)

2) DHRUV MITTAL (17BCE2110)

SLOT: G2 / L19+20

DATE: 31-03-2019

SUBJECT: DATA MINING (FINAL REVIEW-3)

SUBMITTED TO : PROF. LOKESH KUMAR

PROJECT TITLE

WEATHER FORECAST USING DATA MINING TECHNIQUES

ABSTRACT

Our project is titled “WEATHER FORECAST USING DATA MINING TECHNIQUES”, In this we usually consider large dataset like nearly 50-60 years of data and will predict the future weather depending on the decision tree we obtain. The decision tree is trained using the dataset of the historical weather condition of Delhi region, hourly readings. This study makes a mention of various techniques and algorithms that are likely to be chosen for weather prediction and highlights the performance analysis of these algorithms. Various attributes like temperature, humidity, pressure, etc are used.

SCOPE OF THE PROJECT

The weather prediction is mostly based on the physical changes observed and some statistics. As we have a huge amount of data producing everyday regarding weather. We can use that to get some useful inference from it. That data can be used to predict the future weather condition.

INTRODUCTION

Weather forecasting is especially involved with the prediction of weather within the given future time. Weather forecasts provide critical information about future weather. The prediction of weather is important for numerous applications. Some of them are climate watching, drought detection, severe weather prediction, agriculture and production, planning in energy industry, aviation industry, communication, pollution dispersal, and so forth.

Objective: Collect dataset from Kaggle and include necessary attributes required for the consideration of predicting the weather

Data set used: Delhi weather data after data cleaning is done.

Data mining technique used: decision tree, J48 algorithm and naïve bayes

DECISION TREE AND J48 ALGORITHM FOR WETHER PREDICTION

Decision Tree: A decision tree is a predictive machine-learning model that decides the target value (dependent variable) of a new sample based on various attribute values of the available data. The internal nodes of a decision tree denote the different attributes, the branches between the nodes tell us the possible values that these attributes can have in the observed samples, while the terminal nodes tell us the final value (classification) of the dependent variable.

J48 Algorithm: The J48 Decision tree classifier follows the following simple algorithm. In order to classify a new item, it first needs to create a decision tree based on the attribute values of the available training data. So, whenever it encounters a set of items (training set) it identifies the attribute that discriminates the various instances most clearly. This feature that is able to tell us most about the data instances so that we can classify them the best is said to have the highest information gain. Now, among the possible values of this feature, if there is any value for which there is no ambiguity, that is, for which the data instances falling within its category have the same value for the target variable, then we terminate that branch and assign to it the target value that we have obtained.

ATTRIBUTES

1) Condition
2) Dew point measure
3) Fog
4) Hail
5) Humidity
6) Pressure measure
7) Rain
8) Snow
9) Temperature measure
10) Thunder
11) Tornado

KEYWORDS

1) Data Mining
2) Decision Tree
3) Ensemble Technique
4) Pre-Processing
5) Weather Prediction

ATTRIBUTES DESCRIPTION

1) Condition (_conds):

To determine the weather we need to have a data set where certain conditions are to be given.
Eg: Smoke, Fog, Clear, Shallow fog, Scatter cloud, Haze, Mostly Cloudy, Partly Cloudy etc..

So, from this we can come to a conclusion that condition of the weather acts as one of the attribute and primary attribute.

2) Dew point measure (_dewptm):

This measure is for the saturation of water with water vapour and their saturation point is called dew point and they are taken into account as an attribute.

We give numerical value for the calculation of decision tree

3) Fog (_fog):

It consists of visible cloud and water droplets suspended in the air at or near earth's surface.

We determine them in binary in order to reduce the complication.
If fog exists then we give '1' as input or else we provide '0' as input.

4) Hail (_hail):

Pallets of frozen rain which fall in showers from cumulonimbus clouds.
We consider this in binary too.
If hail is present we denote as '1' else we'll provide it with '0'.

5) Humidity (_hum):

It determines the content of water vapour present in the atmosphere.
Levels of humidity is given in numerical value.

6) Pressure measure (_pressure):

It determines the barometric pressure in the atmosphere and the values are only present in positive numbers rather than negative numbers. After thorough data cleaning negative numbers are cleared from the dataset.

7) Rain (_rain):

In the same way done for fog and hail we consider binary if the condition is rain then we denote it with'1' and '0' for negative.

8) Snow (_snow):

Since this is also taken in binary it will be easy to distinguish between snow and fog. So, binary representation is considered here too with '1' as snow included and '0' for snow not included.

9) Temperature measure (_tempm):

The temperature measure is done by required numerical value which is one of the attribute required to determine the weather forecasting.

10) Thunder (_thunder):

This can be considered in binary notation as '1' if a thunder is recorded and else '0' is used instead.

11) Tornado (_tornado):

This also considered the same case as of the thunder which is binary where '1' is used for recording of tornado else '0' is returned in the dataset.

DECISION TREE IMPLEMENTATION FOR WEATHER FORECASTING

PYTHON IMPLEMENTATION:

```
import csv
import pandas as pd
import numpy as py
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt


dataset = pd.read_csv('D:\KRUPAL\Weather.csv')



print("Rows and Columns\n")
print(dataset.shape)


#temperature outliners

print("\n\nThe plot of temperature variation with outliers\n\n")
plt.plot(dataset[' _tempm'])
plt.title('Temperature with outliners\n')
plt.show()


print("\nRemoving the tuple with more than 1.5*(75%ile Quartile) value of temp\n")
w=dataset[' _tempm'].quantile(.75)*(1.5)
```

```
print(w," is the 1.5*75%ile\n")


print("\n below is the plot after removing outliners\n")
dataset = dataset[dataset[' _tempm'] < w]



plt.plot(dataset[' _tempm'])
plt.title('Temperature without outliners\n')
plt.show()


#pressure ouliners

print("\n\nThe plot of pressure variation with outliers\n\n")
plt.plot(dataset[' _pressurem'])
plt.title('Pressure with outliners\n')
plt.show()


print("\nRemoving the tuple with more than 1030 and less than 880 value of pressure\n")
w=1030

print("\n below is the plot after removing outliners\n")
dataset = dataset[dataset[' _pressurem'] < w]
dataset = dataset[dataset[' _pressurem'] >880]


plt.plot(dataset[' _pressurem'])
plt.title('Pressure without outliners\n')
plt.show()


#humidity ouliners

print("\n\nThe plot of humidty variation with outliers\n\n")
```

```
plt.plot(dataset['_hum'])
plt.title('Humidity with outliners\n')
plt.show()


print("\nRemoving the tuple with more than 1.5*(75%ile) of humidity values\n")
w=dataset['_hum'].quantile(.75)*(1.5)

print("\n below is the plot after removing outliners\n")
dataset = dataset[dataset['_hum'] < w]


plt.plot(dataset['_hum'])
plt.title('Humidity without outliners\n')
plt.show()



print("\n\nanalysis of the first 5 datapoints\n")
print(dataset.head())

print("\nThe target attributes\n")
for i in dataset['_conds'].unique():
        print(i)


x=dataset.drop('_conds',axis=1)
y=dataset['_conds']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20)

classifier = DecisionTreeClassifier()

classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)
```

```
print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test, y_pred))

print(classifier.predict([[9,0,0,27,1010,0,0,30,0,0]]))

print(classifier.predict([[0,0,0,60,1020,0,0,30,1,0]]))

print(classifier.predict([[1,1,1,60,1020,1,1,40,1,1]]))

print(classifier.predict([[1,1,1,1,1000,1,1,40,1,1]]))
```

IMPLEMENTATION

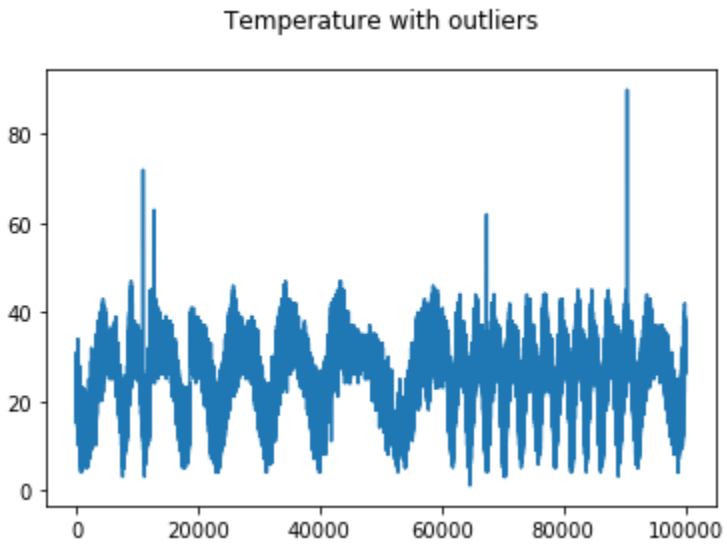
```
Rows and Columns

(99958, 11)
```

In this we first take the data set and there undergone data cleaning so that plotting the graph more specified and understandable.

After the data cleaning we implemented the individual dataset accordingly and plotted their variation according to the year respectively.

This implementation is based for Temperature:

The following plotted graphs are:

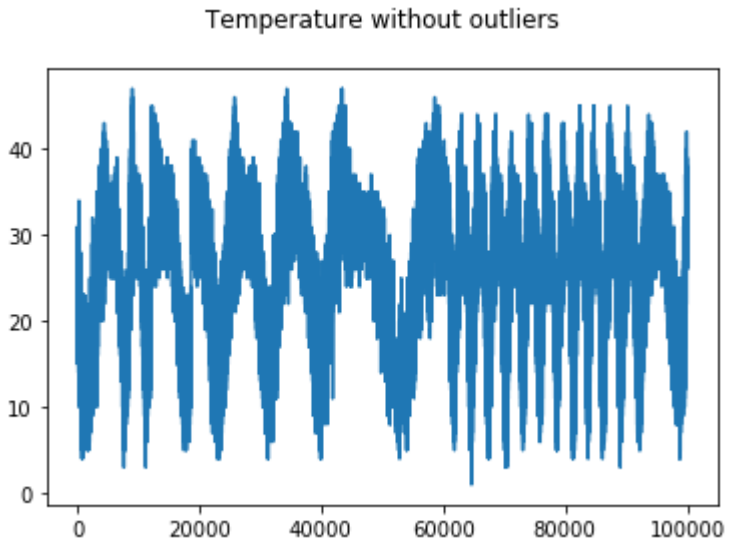



```
Removing the tuple with more than 1.5*(75%ile Quartile)

value of temp 48.0 is the 1.5*75%ile
```

This is with outliers as mentioned above and as we can see the highest temperature ever recorded here is nearly 90 degrees which seems absurd.

In order to resolve this problem we need to remove the outliers so we used quartile division (75%) and multiplied with 1.5 so that above that quartile outliers are removed.
Here is a picture depicting without outliers.

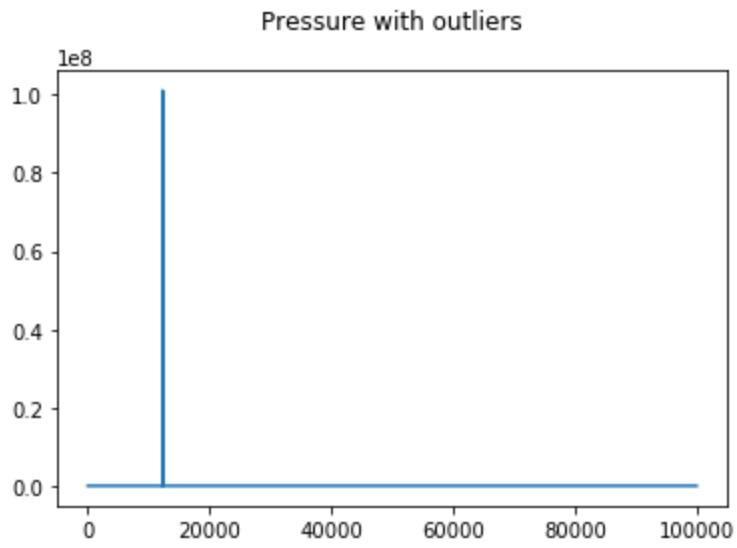
The temperature varies hourly from 1996 to 2017.



This project, is fully functional and reliable application that has been worked upon by our group. In the above report it can be easily spot the data set used, Procedure of Implementation and Working, Code has been significantly displayed.

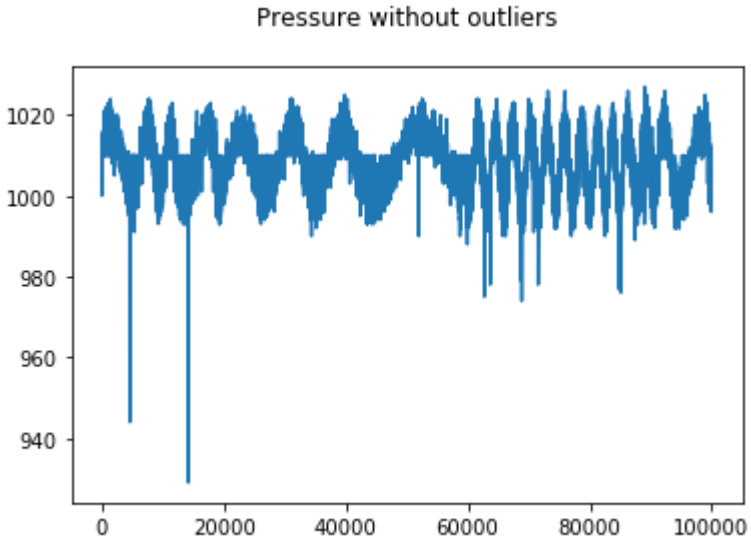
The implementation for Pressure:

In this we first plotted the graph with outliers



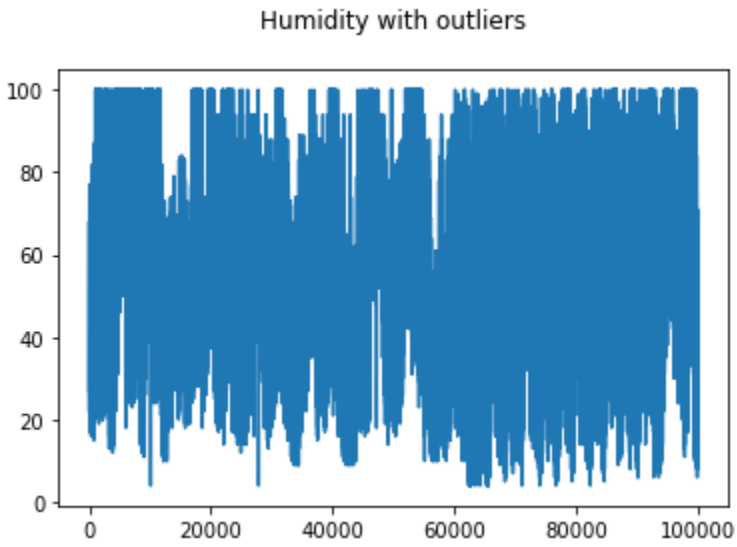

```
Removing the tuple with more than 1030 and less than 880
value of pressure
```

After the data cleaning the output is as followed:



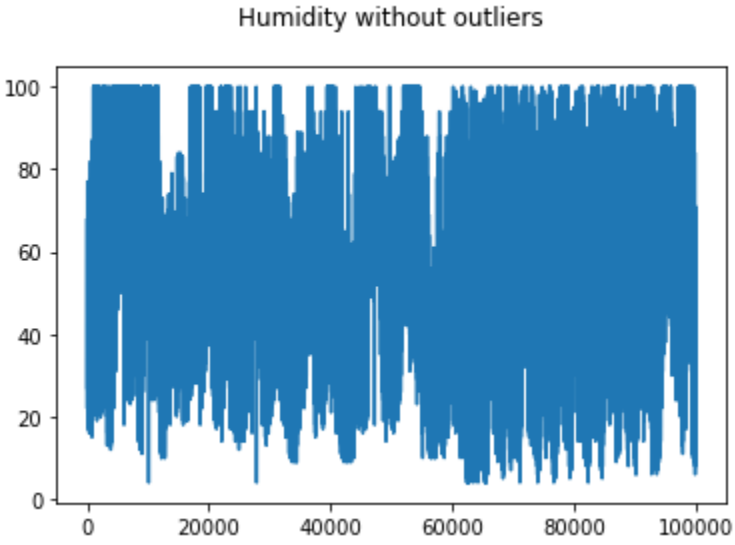
In the same way with humidity:

The plotted graph is with outliers



Removing the tuple with more than 1.5*(75%ile) of humidity values

The plotted graph without outliers:



Analysis of the first 5 datapoints:

analysis of the first 5 datapoints

	_conds	_dewptm	_fog	_hail	_hum	_pressurem	_rain	_snow	_tempm	\
0	Smoke	9	0	0	27	1010.0	0	0	30	
1	Smoke	10	0	0	32	1000.0	0	0	28	
2	Smoke	11	0	0	44	1000.0	0	0	24	
3	Smoke	10	0	0	41	1010.0	0	0	24	
4	Smoke	11	0	0	47	1011.0	0	0	23	

	_thunder	_tornado
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

The target attributes:

Smoke
Clear
Haze
Unknown
Scattered Clouds
Shallow Fog
Mostly Cloudy
Fog
Partly Cloudy
Patches of Fog
Thunderstorms and Rain
Overcast
Rain
Light Rain
Light Drizzle
Drizzle
Mist
Thunderstorm
Light Thunderstorms and Rain
Light Thunderstorm
Squalls

```
Heavy Rain
Light Haze
Widespread Dust
Funnel Cloud
Heavy Thunderstorms and Rain
Heavy Thunderstorms with Hail
Light Rain Showers
Volcanic Ash
Thunderstorms with Hail
Partial Fog
Light Fog
Heavy Fog
Blowing Sand
Sandstorm
Light Hail Showers
Light Sandstorm
Rain Showers
```

```
[[  6   0   0 ...   0   0   9]
 [  0 213   1 ...   0   1  16]
 [  0   0   0 ...   0   0   0]
 ...
 [  0   0   0 ...   1   0   0]
 [  0  14   0 ...   0   7   4]
 [ 18  32   0 ...   0   3 141]]
```

Following are the target values with their recall, f1-score and support values.

	precision	recall	f1-score	support
Blowing Sand	0.06	0.10	0.08	62
Clear	0.30	0.33	0.32	643
Drizzle	0.00	0.00	0.00	19
Fog	0.67	0.78	0.72	546
Funnel Cloud	0.00	0.00	0.00	1
Haze	0.70	0.78	0.74	9411
Heavy Fog	0.22	0.26	0.24	69
Heavy Rain	0.08	0.33	0.13	3
Heavy Thunderstorms and Rain	0.00	0.00	0.00	4
Light Drizzle	0.15	0.10	0.12	78
Light Fog	0.12	0.07	0.09	14
Light Haze	0.00	0.00	0.00	0
Light Rain	0.69	0.69	0.69	265
Light Rain Showers	0.00	0.00	0.00	2
Light Sandstorm	0.00	0.00	0.00	0
Light Thunderstorm	0.57	0.27	0.36	15
Light Thunderstorms and Rain	0.40	0.38	0.39	45
Mist	0.79	0.75	0.77	1799
Mostly Cloudy	0.26	0.11	0.16	338
Overcast	0.12	0.06	0.08	49
Partial Fog	0.45	0.38	0.41	222
Partly Cloudy	0.17	0.09	0.12	406
Patches of Fog	0.44	0.34	0.38	197
Rain	0.41	0.33	0.37	90
Rain Showers	0.00	0.00	0.00	0
Sandstorm	0.00	0.00	0.00	1
Scattered Clouds	0.17	0.05	0.08	455
Shallow Fog	0.58	0.56	0.57	352
Smoke	0.63	0.61	0.62	4103
Squalls	0.00	0.00	0.00	2
Thunderstorm	0.80	0.70	0.75	40
Thunderstorms and Rain	0.69	0.69	0.69	88
Thunderstorms with Hail	1.00	1.00	1.00	1
Unknown	0.32	0.12	0.17	58
Widespread Dust	0.33	0.24	0.28	582

	precision	recall	f1-score	support
Blowing Sand	0.05	0.07	0.06	76
Clear	0.30	0.33	0.31	646
Drizzle	0.00	0.00	0.00	20
Fog	0.67	0.82	0.73	560
Haze	0.70	0.77	0.73	9353
Heavy Fog	0.28	0.20	0.24	94
Heavy Rain	0.00	0.00	0.00	4
Heavy Thunderstorms and Rain	0.00	0.00	0.00	3
Light Drizzle	0.17	0.15	0.16	65
Light Fog	0.00	0.00	0.00	15
Light Rain	0.69	0.71	0.70	279
Light Rain Showers	0.00	0.00	0.00	0
Light Sandstorm	0.00	0.00	0.00	2
Light Thunderstorm	0.21	0.31	0.25	13
Light Thunderstorms and Rain	0.19	0.29	0.23	21
Mist	0.79	0.72	0.75	1858
Mostly Cloudy	0.19	0.10	0.13	279
Overcast	0.17	0.08	0.11	50
Partial Fog	0.37	0.38	0.38	199
Partly Cloudy	0.20	0.11	0.15	410
Patches of Fog	0.37	0.28	0.32	187
Rain	0.31	0.17	0.22	92
Scattered Clouds	0.20	0.08	0.11	412
Shallow Fog	0.65	0.55	0.59	385
Smoke	0.63	0.62	0.63	4170
Squalls	0.00	0.00	0.00	1
Thunderstorm	0.76	0.44	0.56	50
Thunderstorms and Rain	0.81	0.74	0.77	92
Thunderstorms with Hail	1.00	0.50	0.67	4
Unknown	0.38	0.12	0.19	65
Volcanic Ash	0.00	0.00	0.00	1
Widespread Dust	0.31	0.26	0.28	554
avg / total	0.62	0.64	0.63	19960

micro avg	0.64	0.64	0.64	19960
macro avg	0.32	0.29	0.29	19960
weighted avg	0.62	0.64	0.63	19960

```
['Haze'] ['Haze'] ['Fog'] ['Thunderstorms and Rain']
```

On giving below inputs in the predictor, we get the most likely weather condtion:

```
print(classifier.predict([[9,0,0,27,1010,0,0,30,0,0]]))
```
['Haze']

```
print(classifier.predict([[0,0,0,60,1020,0,0,30,1,0]]))
```
['Haze']

```
print(classifier.predict([[1,1,1,60,1020,1,1,40,1,1]]))
```
['Fog']

```
print(classifier.predict([[1,1,1,1,1000,1,1,40,1,1]]))
```
['Thunderstorms and Rain']

NAÏVE BAYES APPROACH FOR WETHER PREDICTION

Naïve Bayes: It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter.

Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

The Naïve Bayes algorithm is a simple probabilistic classifier that calculates a collection of probabilities by investigating frequency and combination of values in a given data set. The algorithm is based on applying Bayes theorem with the "naïve" assumption of independence between every pair of features. Due to simple structure of Naive Bayes, construction of it is very simple and also has several advantages. Moreover, the inference (classification) is achieved in a linear time (while the inference in Bayes networks with a general structure is known to be NP-complete). Also, it does not require much larger data set smaller data set can also be used. Finally, the construction of naive Bayes is incremental, in the sense that it can be easily updated (namely, it is always easy to consider and take into account new cases in hand). Suppose Ci be diabetes risk group i and N be input variables that are used in a model and under the assumption of all variables are independent. To predict a class of diabetes risk, a model of Naive Bayes can be defined by

$$P(C_i|\,N) \quad = \quad \frac{P(N\,|C_i) \;\times\; P(C_i)}{P(N)}$$

Where (|) is a posterior probability of a training data set

NAIVE BAYES IMPLEMENTATION FOR WEATHER FORECASTING

PYTHON IMPLEMENTATION:

```
import csv
import random
import math

def loadCsv(filename):
	lines = csv.reader(open(filename, "rb"))
	dataset = list(lines)
	for i in range(len(dataset)):
		dataset[i] = [float(x) for x in dataset[i]]
	return dataset

def splitDataset(dataset, splitRatio):
	trainSize = int(len(dataset) * splitRatio)
	trainSet = []
	copy = list(dataset)
	while len(trainSet) < trainSize:
		index = random.randrange(len(copy))
		trainSet.append(copy.pop(index))
	return [trainSet, copy]

def separateByClass(dataset):
	separated = {}
	for i in range(len(dataset)):
		vector = dataset[i]
		if (vector[-1] not in separated):
			separated[vector[-1]] = []
		separated[vector[-1]].append(vector)
	return separated

def mean(numbers):
```

```
        return sum(numbers)/float(len(numbers))

def stdev(numbers):
        avg = mean(numbers)
        variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
        return math.sqrt(variance)

def summarize(dataset):
        summaries = [(mean(attribute), stdev(attribute)) for attribute in
zip(*dataset)]
        del summaries[-1]
        return summaries

def summarizeByClass(dataset):
        separated = separateByClass(dataset)
        summaries = {}
        for classValue, instances in separated.iteritems():
                summaries[classValue] = summarize(instances)
        return summaries

def calculateProbability(x, mean, stdev):
        exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
        return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def calculateClassProbabilities(summaries, inputVector):
        probabilities = {}
        for classValue, classSummaries in summaries.iteritems():
                probabilities[classValue] = 1
                for i in range(len(classSummaries)):
                        mean, stdev = classSummaries[i]
                        x = inputVector[i]
                        probabilities[classValue] *= calculateProbability(x, mean,
stdev)
        return probabilities

def predict(summaries, inputVector):
        probabilities = calculateClassProbabilities(summaries, inputVector)
        bestLabel, bestProb = None, -1
        for classValue, probability in probabilities.iteritems():
                if bestLabel is None or probability > bestProb:
```

```
                bestProb = probability
                bestLabel = classValue
        return bestLabel

def getPredictions(summaries, testSet):
        predictions = []
        for i in range(len(testSet)):
                result = predict(summaries, testSet[i])
                predictions.append(result)
        return predictions

def getAccuracy(testSet, predictions):
        correct = 0
        for i in range(len(testSet)):
                if testSet[i][-1] == predictions[i]:
                        correct += 1
        return (correct/float(len(testSet))) * 100.0

def main():
        filename = 'pima-indians-diabetes.data.csv'
        splitRatio = 0.67
        dataset = loadCsv(filename)
        trainingSet, testSet = splitDataset(dataset, splitRatio)
        print('Split {0} rows into train={1} and test={2} rows').format(len(dataset),
len(trainingSet), len(testSet))
        # prepare model
        summaries = summarizeByClass(trainingSet)
        # test model
        predictions = getPredictions(summaries, testSet)
        accuracy = getAccuracy(testSet, predictions)
        print('Accuracy: {0}%').format(accuracy)

main()
```

DATA-SET FOR NAÏVE BAYES APPROACH:

	Outlook	Temp	Humidity	Windy
2	Rainy	Hot	High	f
3	Rainy	Hot	High	t
4	Overcast	Hot	High	f
5	Sunny	Mild	High	f
6	Sunny	Cool	Normal	f
7	Sunny	Cool	Normal	t
8	Overcast	Cool	Normal	t
9	Rainy	Mild	High	f
10	Rainy	Cool	Normal	f
11	Sunny	Mild	Normal	f
12	Rainy	Mild	Normal	t
13	Overcast	Mild	High	t
14	Overcast	Hot	Normal	f
15	Sunny	Mild	High	t

-
-
-
-
-
-
-
-

Pros:

- It is easy and fast to predict class of test data set. It also perform well in multi class prediction
- When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
- It perform well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

Cons:

- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency". To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.

- On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.

- Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

DECESION TREE V/S NAÏVE BAYES

Decision Trees: A decision tree is a predictive machine-learning model that decides the target value (dependent variable) of a new sample based on various attribute values of the available data. The internal nodes of a decision tree denote the different attributes, the branches between the nodes tell us the possible values that these attributes can have in the observed samples, while the terminal nodes tell us the final value (classification) of the dependent variable.

Naive Bayes: It is type of supervised learning algorithm. It assumes a underlying probabilistic model (Bayes theorem). You can look at more detail about this algorithm here . It is majorly used when more number of classes to predict like Text Classification, Spam Filtering, Recommendation System and others.

- It is easy to implement
- Requires small amount of data to train model
- Good results obtained in most of the cases

- **Naïve Bayes:**

    1. Work well with small dataset compared to DT which need more data
    2. Lesser overfitting
    3. Smaller in size and faster in processing

- **Decision Tree:**

    1. Decision Trees are very flexible, easy to understand, and easy to debug
    2. No pre-processing or transformation of features required
    3. Prone to overfitting but you can user pruning or Random forests to avoid that.

REVIEW OF LITERATURE BACKGROUND

In order to predict future weather or coming weather the system needed to require input the simply patterns of the weather, based on the observation and transitional patterns the system generate future attainable pattern of weather. Currently one of the most widely used techniques for weather prediction is data mining. Data mining offers a way of analysing data statistically and extract or derive such rules that can be used for predictions. Presently it is being used in many domains such as stock market, sports, banking section, etc. Scientists have now realized that data mining can be used as a tool for weather prediction as well. The basic entity of data mining is data itself. It is defined as raw set of information which can be used to extract meaningful information depending upon the requirements of the application.

FUTURE USE

There is a future use in predicting the weather beforehand and also to compute humidity and pressure simultaneously using the attributes mentioned above particularly. This will enhance our possibility to predict the weather accurately and precisely which will be a major advantage in future.

REFERENCES

Paper: https://www.ijedr.org/papers/IJEDR1702035.pdf

Paper: http://www.mecs-press.org/ijieeb/ijieeb-v4-n1/IJIEEB-V4-N1-7.pdf

Paper: http://www.indjst.org/index.php/indjst/article/viewFile/101962/74214